Mathematical Biosciences and Engineering

MBE, 18 (1): 132–153.
DOI: 10.3934/mbe.2021007
Received: 28 August 2020
Accepted: 12 November 2020
Published: 23 November 2020

http://www.aimspress.com/journal/MBE

Research article

Wearable on-device deep learning system for hand gesture recognition based on FPGA accelerator

Weibin Jiang[1], Xuelin Ye[2], Ruiqi Chen[1,3], Feng Su[3,4], Mengru Lin[1], Yuhanxiao Ma[5], Yanxiang Zhu[3] and Shizhen Huang[1,*]

[1] College of Physics and Information Engineering, Fuzhou University, Fuzhou 350116, China
[2] Department of Statistics, University of Warwick CV4 7AL, United Kingdom
[3] VeriMake Research, Nanjing Qujike Info-tech Co., Ltd., Nanjing 210088, China
[4] Tsinghua-Berkeley Shenzhen institute, Tsinghua University, Shenzhen 518055, China
[5] Gallatin School of Individualized Study, New York University, NY 10012, United States

* **Correspondence:** Email: hs501@fzu.edu.cn.

Abstract: Gesture recognition is critical in the field of Human-Computer Interaction, especially in healthcare, rehabilitation, sign language translation, etc. Conventionally, the gesture recognition data collected by the inertial measurement unit (IMU) sensors is relayed to the cloud or a remote device with higher computing power to train models. However, it is not convenient for remote follow-up treatment of movement rehabilitation training. In this paper, based on a field-programmable gate array (FPGA) accelerator and the Cortex-M0 IP core, we propose a wearable deep learning system that is capable of locally processing data on the end device. With a pre-stage processing module and serial-parallel hybrid method, the device is of low-power and low-latency at the micro control unit (MCU) level, however, it meets or exceeds the performance of single board computers (SBC). For example, its performance is more than twice as much of Cortex-A53 (which is usually used in Raspberry Pi). Moreover, a convolutional neural network (CNN) and a multilayer perceptron neural network (NN) is used in the recognition model to extract features and classify gestures, which helps achieve a high recognition accuracy at 97%. Finally, this paper offers a software-hardware co-design method that is worth referencing for the design of edge devices in other scenarios.

Keywords: micro-control unit (MCU); accelerator; inertial measurement unit (IMU); field-programmable gate array (FPGA); gesture recognition; convolutional neural network (CNN)

1. Introduction

Gesture recognition has attracted much attention across different fields due to its various applications in healthcare, rehabilitation, caring for the deaf-mutes, etc. [1]. Through our field trip to a Chinese provincial medical institute, we got to know that it is normally hard for doctors to measure the effectiveness of rehabilitation follow-up treatment after patients leave the hospital. Therefore, we are commissioned by this institute to design an offline wearable device for real-time gesture recognition to help with remote diagnosis. In the field of hand gesture recognition, we laid the groundwork for the design of an edge device that is able to undergo on-device AI computing. In other words, with our efficient hardware, robust software, and improved AI algorithms, we are able to process data and complete the recognition process locally, rather than in the cloud or on PC.

In the field of gesture recognition, image processing has been a popular established protocol [2,3], and the use of artificial intelligence (AI) has dramatically improved the accuracy and robustness of image-based gesture recognition [4,5]. However, it usually requires a powerful processor for completing various functions, which severely limits the portability of the device. Moreover, image-processing technology may be interfered with by multiple factors, including ray and skin color, which significantly raises training costs and may potentially bring ethical issues. Furthermore, image-processing technology cannot efficiently ascertain the speed during gesture recognition, making it hard for its application in similar medical rehabilitation scenarios.

In terms of sensors for gesture recognition, surface electromyography (SEMG) sensor [6], highly flexible fabric strain sensor [7], and many other sensors are limited by their higher costs and power consumption. That is why we decide to use the inertial measurement unit (IMU) sensors. With its characteristics of low cost, low power consumption, and high performance, it is an excellent solution to record the acceleration and angular velocity of gestures.

To date, in existing gesture recognition systems to our knowledge, gesture data are collected and then sent to a remote processing system with higher computing power, which significantly increases the costs.

In this paper, a prototype of a low-cost wearable computing system for gesture recognition is proposed, which uses IMU sensors to collect data and locally implements gesture recognition functions, without relaying anything to other devices with higher computing power. As far as we know, it is probably the most effective system that simultaneously achieves low-power, low-latency, and high-accuracy in the field of gesture recognition.

The contributions of this study are as follows: (1) A highly-integrated wearable edge device that locally recognizes hand gestures, rather than uploading data to the cloud as in previous studies; (2) A low-power field-programmable gate array (FPGA) Accelerator deployment via a serial-parallel hybrid method to reduce the resource consumption and improve computational efficiency; (3) A software-hardware co-design based on the Cortex-M0 IP core that improves the system's generalization ability, which is also referable for other scenarios; (4) A new activation function for the NN+CNN structure, which promotes the recognition accuracy; (5) Open-source data for experiments that will be continuously updated.

The rest of the paper is organized as follows. Section 2 introduces the existing gesture recognition methods and AI implementations in wearable designs. The architecture for this software-hardware co-design system, as well as the proposed algorithms for the accelerator, are explained in detail in section 3. Section 4 displays the main part of our work, including the

de-noising process, serial-parallel hybrid method, and resource-efficient scheduling of calculations in the FPGA accelerator. Section 5 demonstrates the datasets, experimental results, and the comparisons of resource consumption and performance between different systems on-chip, including MCUs, SBCs, and desktop processors. Finally, we conclude our work in section 6.

2. Related work

Some common gesture recognition methods and AI implementations in wearable devices are reviewed in the subsequent two paragraphs.

2.1. Gesture recognition methods

The main methods for gesture recognition are image (video) processing and sensor data processing. Visual interest point [8–10], a classic and effective means of gesture recognition, is based on the color, shape, or motion. Neural network algorithms, including three-dimensional (3-D) CNNs, hybrid deep learning, and long short-term memory (LSTM), are widely used in image processing for gesture recognition [11–13]. These approaches effectively reduce the effects of various illuminations, views, and self-structural characteristics. Depth maps have been used to enhance the effect of a large number of small articulations in the hand [14,15]. The IMU or similar sensors are widely used, for instance, in studies where gloves have been used to measure acceleration, finger knuckle angle, and spatial position of hands to recognize gestures [6,16]. In such applications, sensors, including IMU, resistance strain gauge, image sensors, and acceleration sensors are distributed on the back of the hand, forefinger, and the middle finger. Typically, IMU or acceleration sensors are integrated into the peripherals of many wearable devices [17–19]. Previous studies have reported that the values collected by the acceleration sensor are combined with the gestures dictionary, and then feature extraction is used to convert the problem to the classification problem, thus realizing the gesture recognition [20,21]. However, all the proposed methods above used personal computers (PCs) or more powerful devices to process data, which seems redundant and inconvenient for mobile uses such as follow-up treatment for rehabilitation. In contrast, our device can achieve data collection, data processing, and gesture recognition totally offline. Meanwhile, it's worth noting that although RNN and LSTM generally have better performance on time series data, and have been implemented on FPGA in several studies [22–24], they do not contribute to our research objective. They are usually achieved by High Level Synthesis (HLS) instead of hardware description language, which brings uncontrollability to resource and power consumption, reduces the system's portability, and adversely affects future Application-Specific Integrated Circuit (ASIC) deployment. Compared with its drawbacks, the mere improvements in accuracy are not worth mentioning in our case.

2.2. AI implementations in wearable devices

Researchers in [25] designed a CNN accelerator on heterogeneous Cortex-M3/FPGA architecture. The accelerator consists of a convolution circuit and a pooling circuit. This CNN accelerator uses 4901 LUTs without hardware multipliers. A throughput of 6.54 GPOS was achieved. A different study used a very low-power NN accelerator on Micro-semi IGLOO FPGA [26], with a multilayer perceptron NN architecture for classification. The shortest accelerator latency was 1.99 μs and the lowest energy was 34.91 MW under different coupling. Similar findings were drawn in [27],

which proposed a recurrent neural network (RNN) model for IoT (internet of things) devices with an end-to-end authentication system based on breathing acoustics. The advancements of these systems in achieving lower power and lower resource consumption is fundamental to our work. And by combining CNN + multilayer perceptron NN, as well as improving the activation function, we lowered the resource consumption and improved recognition accuracy.

3. System and accelerator design

3.1. System design

The structure of the IMU sensor gesture recognition system designed in this paper is shown in Figure 1. This system consists of an FPGA for the Cortex-M0 IP core implementation, and a glove with IMU sensors driven by Cortex-M0. The data is first sent to the acceleration module by different coupling ways, and then the data processing is accelerated by Cortex-M0.

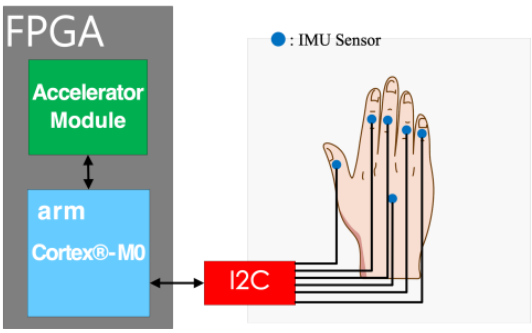


Figure 1. The structure diagram of the proposed gesture recognition system.

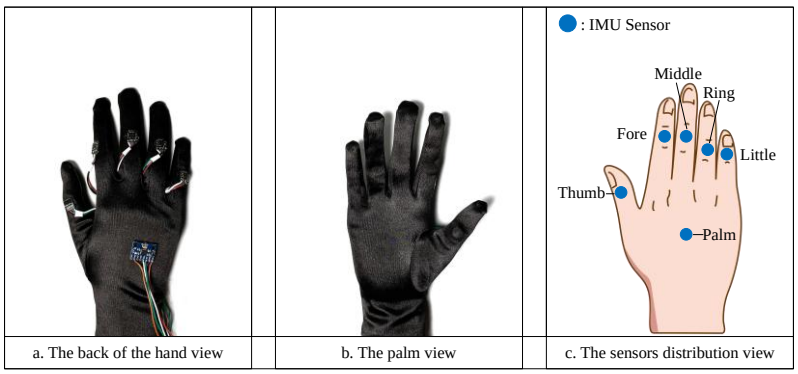


Figure 2. View of the IMU positions on the data glove.

We made two sets of devices, one for the left hand and one for the right hand. The actual right-handed glove prototype with six IMU sensors is shown in Figure 2. There are five sensors distributed on the second joint of each finger, including thumb, fore, middle, ring, and little, as well as one sensor placed on the back of the palm. The sensors are driven, and simultaneously the data is obtained by the Cortex-M0 via the I2C communication protocol. The data collected undergoes Kalman filtering in the inner module. Two-dimensional (2-D) data in the six sensors are collected

and then extracted in the return system for gesture classification. The gestures we sought to recognize are gestures for numbers from one to ten, as demonstrated in Figure 3.

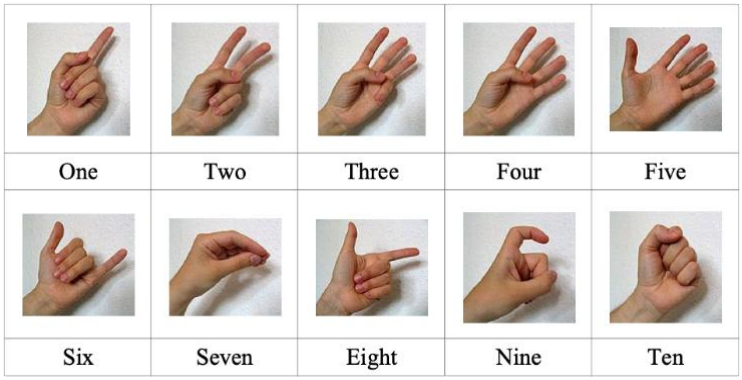


Figure 3. Chinese number gestures.

3.2. Accelerator design

3.2.1. Accelerator architecture

A schematic diagram of the accelerator is shown in Figure 4, including the preprocessing module (PM), feature extraction module (FEM), and classification module (CM).

After the Cortex-M0 drives the IMU, each of the six sensors collects the 2-D angle data with 12 dimensions. The data are then sent to the PM by the Kalman filtering algorithm in the IMU module. The interference caused by external factors such as a slight movement in gesture changes is reduced by smoothing de-noising based on the wavelet transform (WT). The sliding-window and bottom-up (SWAB) algorithm is subsequently used to segment and extract effective data. The extracted effective data are input into the FEM to extract the eigenvalue with CNN. Then the extracted eigenvalues undergo the feature rescaling operation by the Cortex-M0, and are then input into the CM for classification by the multilayer perceptron neural network. Finally, the recognized gesture results are sent to M0 as the output.

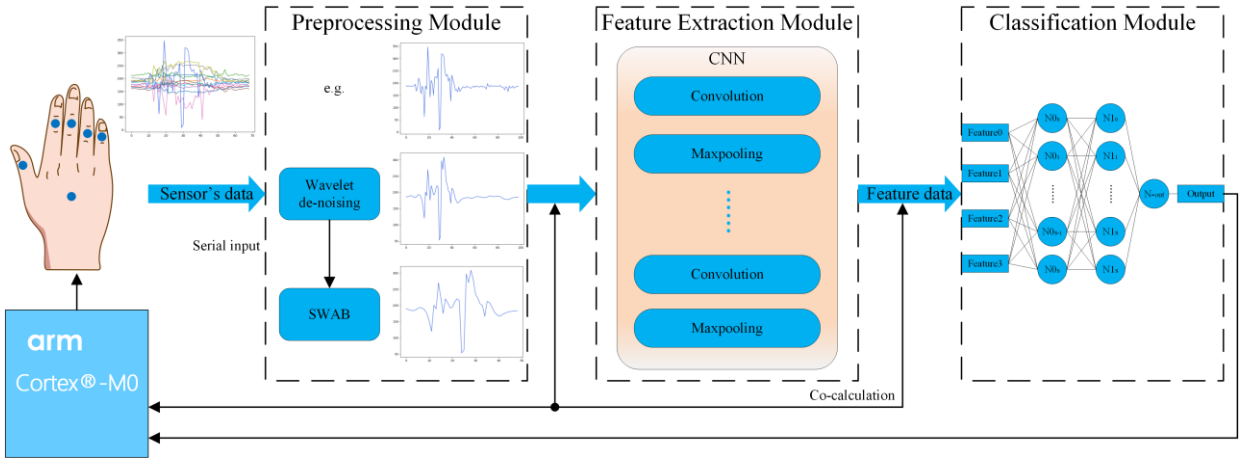


Figure 4. The structure diagram of the accelerator.

Compared to the traditional gesture recognition methods or neural network classifications in the MCU platform, the PM introduced in our approach effectively reduces noise in the downstream processing and the amount of data processed. Additionally, the system does not need expert supervision once trained to properly classify the input data, as CNN is introduced to extract hidden features from the data collected by the IMU sensors [20].

3.2.2. Proposed aalgorithm

As previously mentioned, in order to achieve effective gesture classification, it is necessary to extract various gesture features from the original data. And instead of extracting features by hand, which is highly dependent on expert supervision in the training process [28], we adopted CNN in the system to automatically complete the process. The original signal is preprocessed for feature extraction enhancement. Moreover, the effect of classification is directly affected by the extracted features.

The PM may fall into two classes i.e., the wavelet de-noising and SWAB (the hardware implementation is explained later in section 4.1). During data collection, fatigue or unconscious trembles of the user may introduce noise in the collected gesture data. While noise has large impacts at the stage of feature extraction and lowers the learning rate, de-noising can effectively improve the accuracy of the final model [29–31]. Hence, to overcome this challenge, WT is used for de-noising and filtering [32]. The square-integrable signal x(t) is expressed as

$$WT_x(a,b) = \frac{1}{\sqrt{a}} \int x(t)\psi^* \left(\frac{t-b}{a}\right) dt = \int x(t)\psi^*_{a,b}(t)dt = < x(t), \psi_{a,b}(t) > \quad (1)$$

where b is the time shift, a is the scale factor, and $\psi(t)$ is the basic wavelet. The wavelet basic function is obtained via the shift and stretching of the parent wave as shown in Eq 2.

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi(\frac{t-b}{a}) \quad (2)$$

After passing through the WT, the signals distributed in each layer are uncorrelated to achieve de-noising. As the thresholds of signals at various scales are determined, the thresholds can be quantitatively processed. The corresponding threshold formula is as shown in Eq 3.

$$th_j = \gamma \times \sqrt{\frac{median(|d\_j(k)|)}{0.6754}} \times \frac{\sqrt{2\, ln(length(d_j(k)))}}{ln(j+1)} \quad (3)$$

where γ is the noise power spectrum parameter, median( ) is the median corresponding to the input, $d_j(k)$ is the coefficient of the wavelet in the jth layer, j is the decomposition scale, and $length(d_j(k))$ is the length of the input data. The threshold in the wavelet is the standard for deciding whether to keep or zero the wavelet coefficients. Under high resolution, if the signal amplitude is lower than the threshold value, the wavelet coefficient is set to zero; otherwise, the original wavelet coefficient is retained.

To facilitate quantification, the calculation of FPGA is executed using the fixed threshold formulas shown in Eqs 4 and 5. After de-noising, signals are obtained by reconstruction.

$$th = \gamma \times \sigma \times \sqrt{2\,ln(length(d_j(k)))} \tag{4}$$

$$\sigma = \frac{median(|d_j(k)|)}{0.6754} \tag{5}$$

The process of data collection by the IMU glove includes the preparation stage, movement stage, and completion stage. Regarding signals, values are stable in the preparation stage, but they change in the movement stage before stabilizing again in the completion stage. Some of the data collected in the preparation and completion stage are not essential for sample identification. Such data may severely influence the performance of the subsequent module. Therefore, the filtered data is extracted twice to minimize unnecessary data. Here, this is achieved using the improved SWAB algorithm, which combines the sliding-window (SW) and bottom-up (B-up) algorithms [33]. First, the sum of squares of the 2-D signal values of the collected by the six IMU sensors is taken as the changed measurement as expressed in Eq 6.

$$signal = \sum_{i=1}^{6} (a_{x_i}^2 + a_{y_i}^2) \tag{6}$$

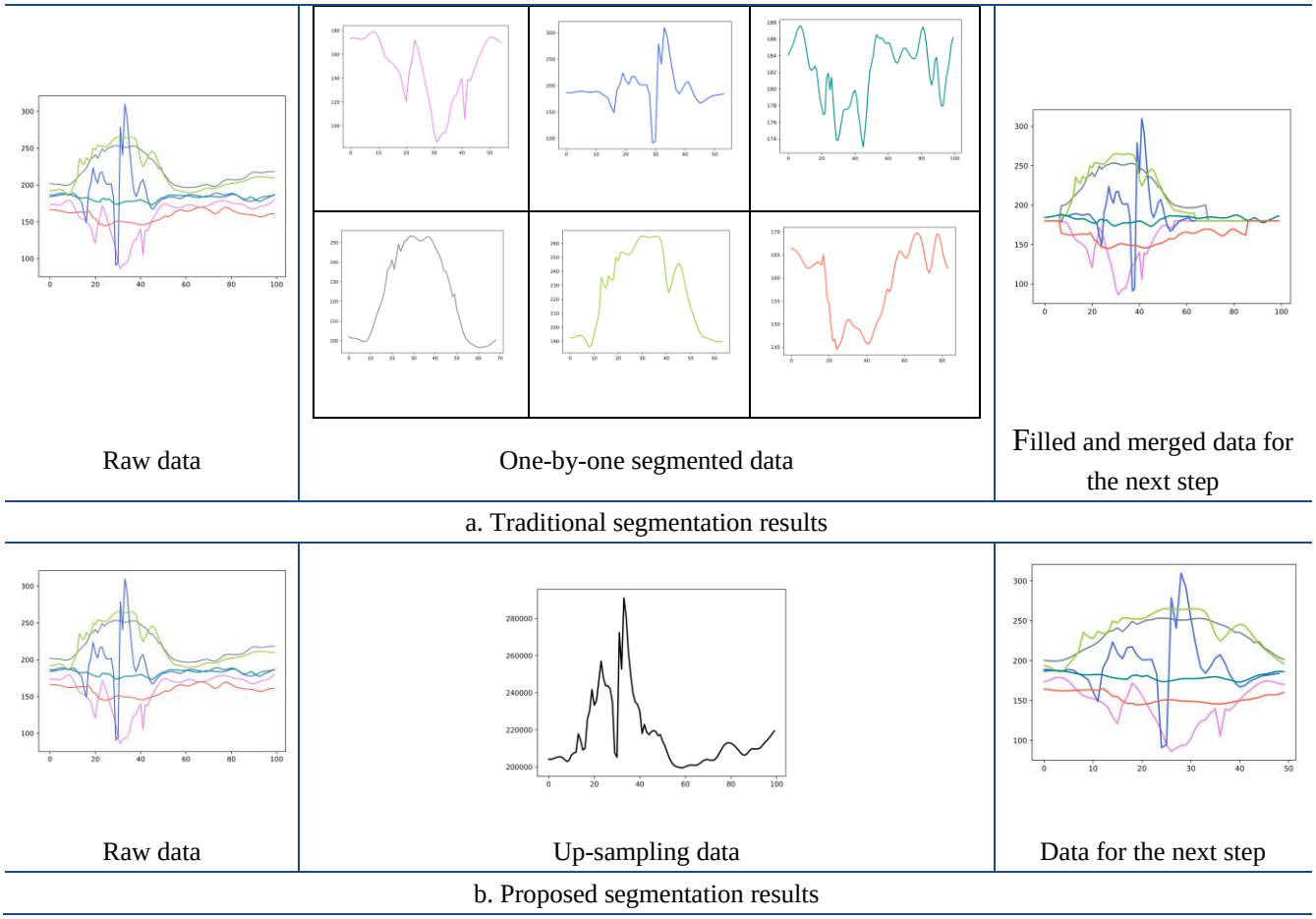
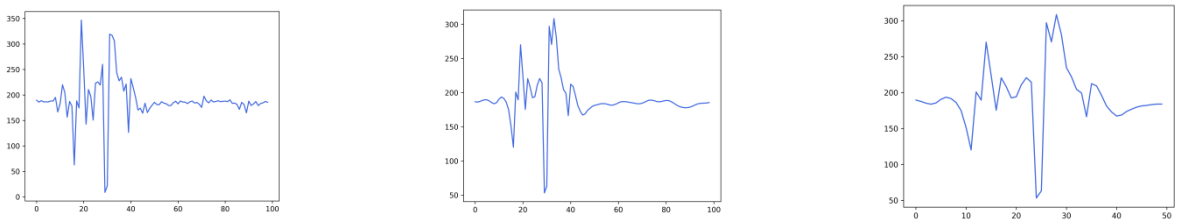


Figure 5. Segmentation results by different SWAB algorithm.

Figure 5a demonstrates the segmentation of valid data in the traditional SWAB algorithm. This algorithm makes the length of feature data varied in different signal channels. However, as it is required to have the same data length in the next stage, other irrelevant and non-contextual data will have to be filled into those short data, which greatly influences the recognition effects.

Figure 5b presents the segmentation in the improved SWAB algorithm. After up-sampling different signal channels, a unified length of valid feature signal is determined and the starting and ending points are passed to the next module. There are three major advantages (a) the feature data lengths are the same after segmentation, (b) the computation required by the filling operation is reduced, (c) the original context of the feature data is preserved.

Signals that can be regarded as the basis for gesture recognition are segmented by setting appropriate thresholds and then passed through the PW. The corresponding waveform is shown in Figure 6.



a. Raw signal b. After wavelet de-noising c. After SWAB

Figure 6. Signal waveform.

Then, in FEM, a CNN model was constructed to extract gesture features (the hardware implementation is explained in section 4.2). After passing signals through PW, these data were used as the bottom input on CNN. Training features were adopted to extract the model. After the training, the model parameters were saved. Finally, the collected gesture data were put into the trained model to extract gesture features for output. A schematic diagram of the CNN structure is shown in Figure 7. This design is composed of three convolutional layers with the number of convolutional kernels similar to those reported in previous studies (4,6,6). Moreover, the size of the convolutional kernel was fixed at 3 × 3. The step size was 1 and the max-pooling was used. The window size was fixed at 2 × 2. The output feature dimension of the full connection layer was 4 and the ReLU (rectified linear activation unit) was selected as the activation function. CNN parameters in each layer are shown in Table 1. The 2-D gesture angle data collected by six sensors, with a size of 50 × 12, were input into the network. To prevent the loss of the boundary data in the process of convolution, the boundary is filled with a numerical value of 180.

Table 1. Parameters of the CNN.

Layer	The number of convolution kernels	Window size	Input	Output
Convolution layer C1	4	3×3	(50, 12, 1)	(50, 12, 4)
Pooling layer P1	/	2×2	(50, 12, 4)	(25, 6, 4)
Convolution layer C2	6	3×3	(25, 6, 4)	(25, 6, 6)
Pooling layer P2	/	2×2	(25, 6, 6)	(12, 3, 6)
Convolution layer C3	6	3×3	(12, 3, 6)	(12, 3, 6)
Pooling layer P3	/	2×2	(12, 3, 6)	(6, 2, 6)

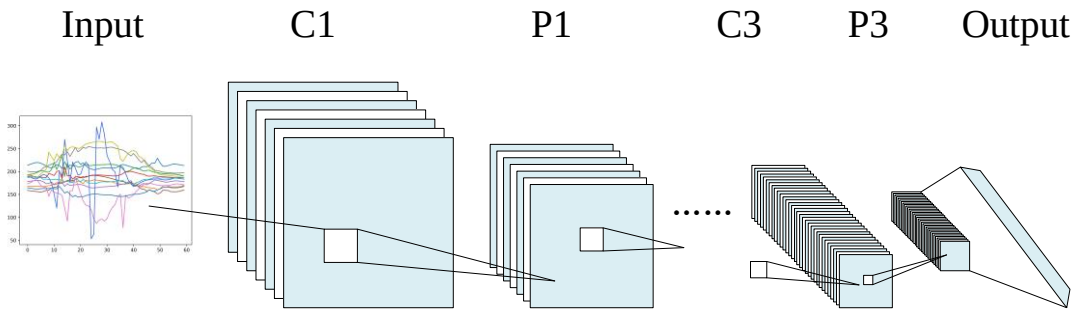


Figure 7. Schematic diagram of the CNN.

1200 sets of original data and 1200 sets of enhanced data from 10 participants comprised of different genders and ages are used for training. The data is described in a more detailed way in section 5 and the original data is publicly available on GitHub and will be updated continuously (the link is provided in SUPPLEMENT). Based on the signal waveforms, the differences in signal results of the different participants corresponded to the differences in the frequency or intensity. Numerically, the differences are in the data continuity and the size of the data. Hence, some methods, such as the frequently used scale, crop, translation, and Gaussian noise [34], increase with image dataset expansion and are effective for the data in this paper.

For FEM and post-stage CM, the input data were normalized to improve the convergence speed of training [35]. The mean normalization formula [36] was used for the FEM and CM inputs with [−1,1] interval as shown in Eq 7.

$$x' = \frac{x - mean\ (x)}{max(x) - min\ (x)} \times 2 \tag{7}$$

The multilayer perceptron NN framework is suited for classification in CM (the hardware implementation is explained in section 4.3). The input and output of NN were set according to the CNN output requirements as shown in Figure 8. Four inputs connected with the FEM output and the final output results corresponded to the recognized gesture number. Next, some parameters of the NN were optimized and different frameworks were designed. They were tested 20 times using different datasets. This analysis revealed that an increase in the number of individual layers or changes in the number of neurons in each NN framework layer had minimal influence on the accuracy [26]. Hence, two hidden layers and eight neurons in each hidden layer were selected.

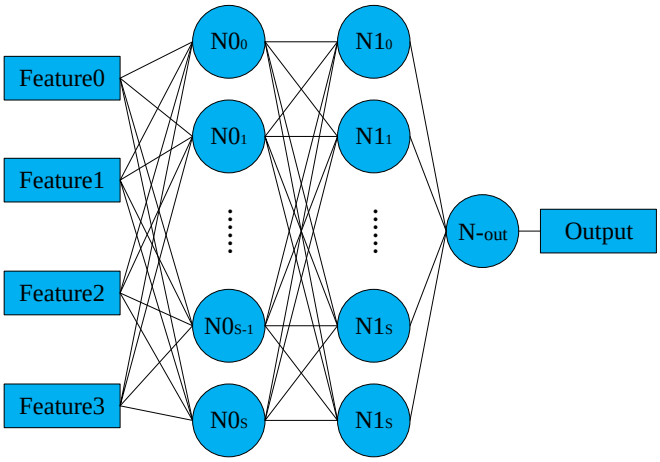


Figure 8. Feedforward neural network architecture.

A new activation function was introduced based on the different layers with different activation functions. Although the traditional ReLU activation function has sparse activation characteristics, the effect of nonlinear fitting data on sparse activation is reduced in the case of a few network layers. Moreover, the response boundary of the ReLU activation function is linear. The ReLU activation function also has a lower fitting ability compared to the Tanh activation function. Thus, ReTanh, a novel activation function, was designed based on the characteristics of the Tanh and ReLU activation functions as shown in Eq 8. The function image is shown in Figure 9.

$$ReTanh = max\ (0, Tanh(x)) \tag{8}$$

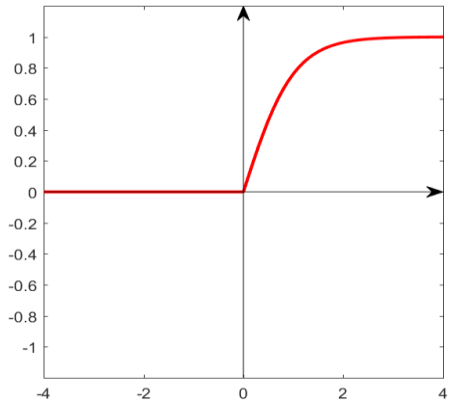


Figure 9. The ReTanh activation function image.

The results of the test accuracy corresponding to the different activation functions used in different layers are shown in Table 2. Note that the last output layer must select ReLU as the activation function. The result shows that the effect of the first layer with the ReLU activation function is poor. This is because, first, the unilateral activation is a linear function, making it hard to fit the complex features under fewer layers. Second, the output range of data cannot be specified twice, resulting in uncontrollable data size and non-convergence in training. Hence, we chose Tanh, ReTanh, and ReLU as the activation functions.

Table 2. Average recognition accuracy for different activation functions.

Activation function		The average accuracy
Hidden Layer 1	Hidden Layer 2	
ReTanh	ReTanh	92.23%
ReTanh	ReLU	93.40%
ReTanh	Tanh	92.59%
Tanh	ReTanh	97.15%
Tanh	Tanh	95.56%
Tanh	ReLU	96.89%
ReLU	ReTanh	89.24%
ReLU	Tanh	91.39%
ReLU	ReLU	87.32%

4. Accelerators implementation

The analysis of the various NN operation modes in the FEM and CM found that the data between network layers are interrelated and that the calculation in the layer only depends on the input data of the previous layer. Thus, each operation unit in the layer is independent. Currently, most neural network operations are based on the central processing unit (CPU). However, the serial computing mode of the CPU cannot give full play to the advantages of a parallel computing network. The development of deep learning has gradually expanded the network structure. For the CPU-based neural network, accelerating the operation by increasing the clock frequency alone would inevitably increase power consumption, but the acceleration effect would not be ideal. Thus, the recently proposed accelerators based on the FPGA, GPU (graphics processing unit), and an ASIC [37–39] should be used to improve the NN performance. The FPGA-based accelerator is more attractive due to its good performance, high-energy efficiency, fast development period, and reconfigurability. Accelerator design can be based on the FPGA hardware and the Cortex-M0 software algorithm introduced in section 3 to minimize resource consumption.

In the following subsections, details about how we simultaneously achieve low-power, low-latency, and high-accuracy on this edge device are explained: (a) how we de-noised data to improve recognition accuracy, (b) how we effectively combine the serial and parallel methods to save resource consumption and improve calculation efficiency, (c) how we schedule the order of calculations to make the best use of the limited resource.

4.1. PM implementation

The introduction of PM in our approach effectively reduces noise in the downstream processing and the amount of data processed, and improves the recognition efficiency of subsequent models. The core of the PM lies in the implementation of the wavelet de-noising, while the SWAB algorithm can be easily implemented as previously described in section 3. The wavelet de-noising is based on the quadrature mirror filter banks. The formulas for calculating the filter coefficient are shown in Eqs 9–12 [40,41].

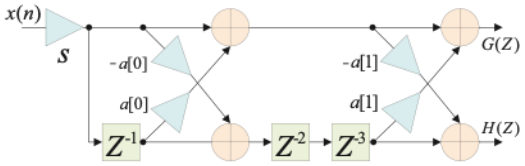
$$G(z) = (1 + a[0]z^{-1} - a[0][1]z^{-2} + a[1]z^{-3})s \tag{9}$$

$$(z) = (-a[1] - a[0]a[1]z^{-1} - a[0]z^{-2} + z^{-3})s \tag{10}$$

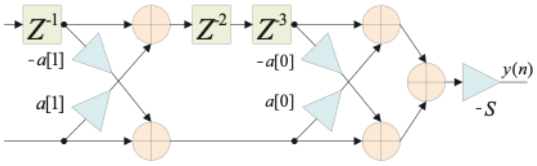
$$\hat{G}(z) = H(-z) \tag{11}$$

$$\hat{H}(z) = -G(-z) \tag{12}$$

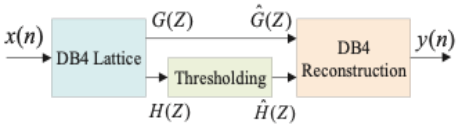
To facilitate the FPGA calculation, s is 0.483, a[0] is 1.732, and a[1] is −0.268 in the equations. Thus, the above structure can be directly realized through the FPGA as shown in Figure 10. The signals are divided into two parts by the filter banks in Figure 10.a. One part with a high frequency-coefficient and the other with a low-frequency coefficient. The noise is reduced by changing the weight of the high-frequency coefficient.



a. DB4 lattice architecture



b. DB4 reconstruction architecture



c. WT DB4 architecture

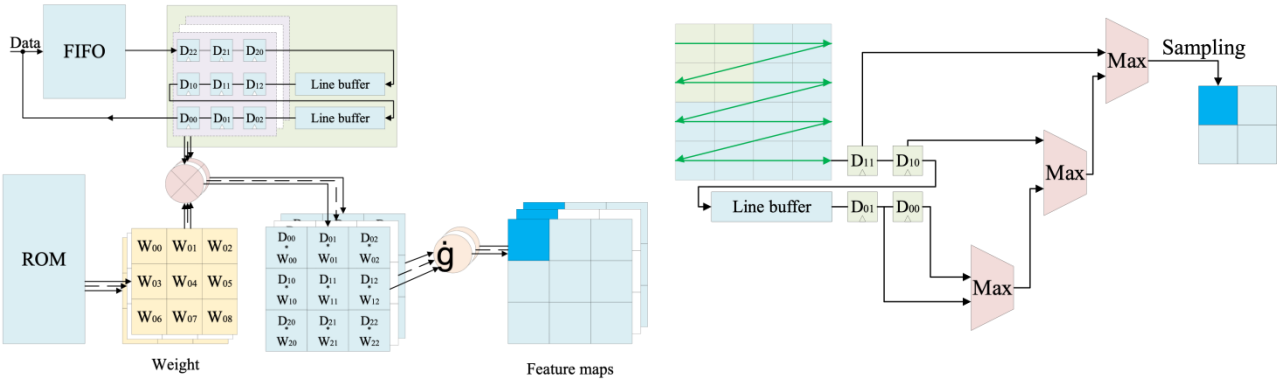
Figure 10. Block diagram of the PM.

4.2. FEM implementation

The main entities in the FEM include convolutions and max-pooling that constitutes the CNN. This part introduces the FPGA hardware implementation of CNN. In the convolution operation, the whole idea is serial input and serial-parallel hybrid calculations. For max-pooling, serial input and parallel calculations are adopted. In this way, resource consumption is saved and calculation efficiency is greatly improved.

Since line-buffer is a common way to image processing [42,43], we proposed register-based line-buffers to implement the convolution operation of 3 × 3 window as shown in Figure 11.a. The weight value corresponding to the current convolution is obtained through the corresponding preset address in the read-only memory (ROM), which allows up to six convolutions to be executed simultaneously. The realization of pooled sampling is similar to convolution implementation. The sample values are obtained systematically through three comparators using the serial data for the local caching as shown in Figure 11.b. These two processes ensure the matching of data transmission, matching of window movement, and performance of each module. This design saves both the data cache capacity and minimizes the consumption of on-chip resources. The

implementation of the fully connected layer is similar to the implementation of the NN in the subsequent CM. This is further discussed in the next section.



a. Block diagram of the convolution operation b. Block diagram of the max-pooling

Figure 11. Block diagram of the FEM.

4.3. CM implementation

The CM is a typical multilayer perceptron with multiple interconnected neurons. While the resource is quite limited, we make the best use of it by dividing and combining the order of calculations of the eight multipliers, in short, we adopted a pipeline-multiplexing structure. This structure is consistent with the fully connected module in the FEM. Regardless of the Tanh or ReTanh function used in the first stage, the corresponding function of the core Tanh is not directly implementable by the FPGA. Here, we used the smoothing interpolation combined with the lookup-table method to fit the activation function [44]. Because the input for the negative Tanh function interval is negative, its absolute value is substituted to the positive interval for calculation. Hence, the negative interval results are the opposite value of the positive interval results. This operation saves the FPGA resources. The function relationship can be fitted by the (n–1) polynomial.

Let $x_j = x^i(j = 1,2,\ldots n)$. The corresponding linear form of a polynomial of n$y = \sum_{i=0}^{n} a_i x_i$ is shown in Eq 13.

$$y = a_0 + \sum_{j=1}^{n} a_j x_j \tag{13}$$

The I = 1, 2, 3…, m experimental points satisfy $x_{ij} = x_i^j$, They are substituted into the Eq 13 and then

$$\begin{cases} ma_0 + \sum_{j=1}^{n}\left(\sum_{i=1}^{m} x_{ij}\right)a_j = \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_{ij}x_{ik}a_0 + \sum_{j=1}^{n}\left(\sum_{i=1}^{m} x_{ij}x_{ik}\right)a_j = \sum_{i=1}^{m} x_{ik}y_i \end{cases} \tag{14}$$

Thus, the polynomial is fitted by the least square method as follows.

$$\sum_{i=1}^{n}\left(\sum_{i=1}^{m} x_i^{j+k}\right)a_j = \sum_{i=1}^{m} x_{ik}y_i k = 0,1 \ldots n \tag{15}$$

The equation is solved thus obtaining $a_0, a_1 \ldots a_n$.

Here, the positive interval is divided into [0,1], (1,2], (2,3], (3,4] and (4,+∞). The corresponding final fitting functions are listed in Table 3. The corresponding absolute error was calculated using MATLAB. The relationship between the fitting function and the original function is drawn by sampling as shown in Figure 12. The data best fit in the interval [0,3]. Moreover, the error is larger when there are more data points. Nevertheless, this does not affect the results since the FPGA hardware calculation is implemented by shifting with 2 decimal places.

Table 3. Fitting formula at different intervals.

Numerical Interval	Formula	Absolute Error
[0,1]	$y = -0.3275x^2 + 1.0977x - 0.0038$	0.0038
(1,2]	$y = -0.1690x^2 + 0.7021x + 0.2324$	0.0039
(2,3]	$y = -0.0282x^2 + 0.1703x + 0.7370$	0.0055
(3,4]	$y = -0.0039x^2 + 0.0313x + 0.9363$	0.0101
(4,+∞)	1	/

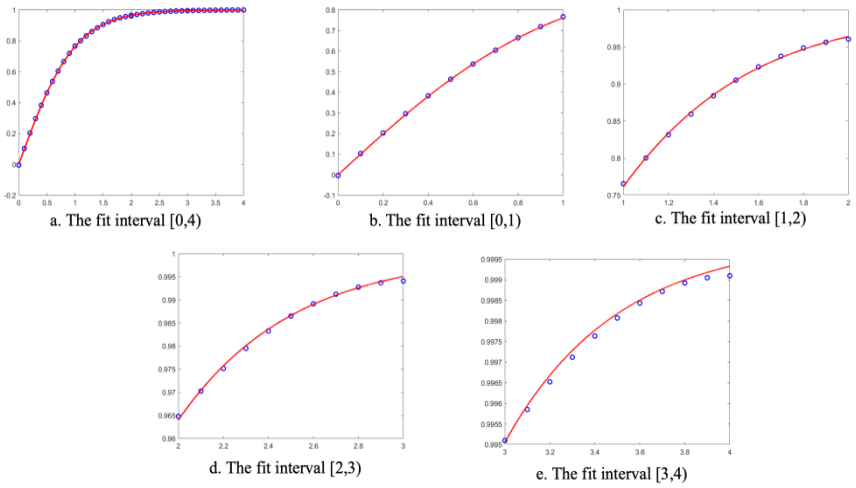


Figure 12. The images of the fitting effect of these formulas.

In this paper, eight multipliers are used in the hidden layer and the processing schedule is demonstrated in Figure 13. The weight and bias in the neurons were input into the corresponding neurons by the external storage for calculation. The data from each module of the previous level were serially input, while the entire neural network was still in a pipeline structure. For the eight multipliers in the hidden layer, operation with the four inputs in each neuron can be conducted simultaneously, except for the three multipliers needed by the activation functions in the first two layers. To complete the calculations in the whole neuron, multiple operation periods are needed in the hidden and output layer. To solve this, a cache should be used to store data between layers. It is also worth noting that, the whole operation was completed in the first hidden layer after eight cycles. For the second layer, it takes eight cycles to complete the operation in each neuron. Thereafter, all the multipliers are used to calculate the activation functions. The operation was completed in three cycles. The whole process takes 21 cycles.

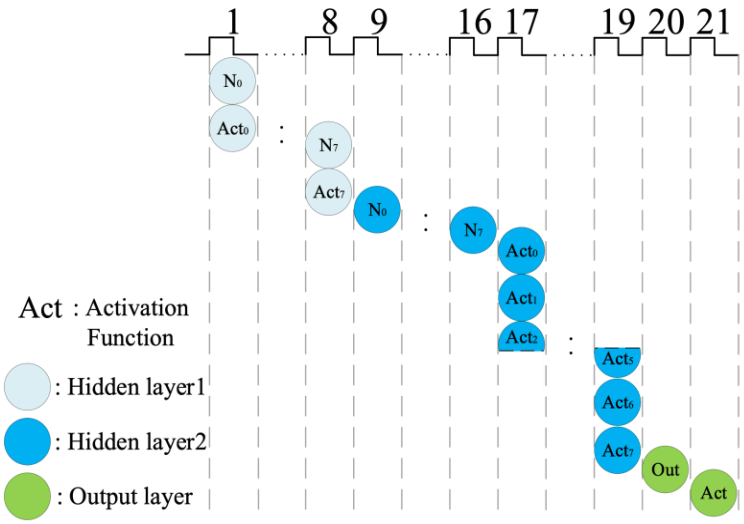


Figure 13. Scheduling of the NN processing.

5. Experimental results and performance analysis

In this section, experiments and discussions are conducted as evidence to support the superiority of the suggested system. In the following sub-sections, we first describe the datasets for experimentation, and then compare our algorithm with other past algorithms that can also be implemented on MCU-level platforms [45]. Finally, we focus on evaluating resource consumption, time performance, and power consumption on different hardware development platforms.

5.1. System datasets

The algorithm proposed for the accelerator was first implemented and tested using python language to validate the concept with our database [46]. This dataset contains: (a) 1200 sets of original anonymous data of 10 different gestures, which are proudly offered by 10 volunteers, and (b) 1200 sets of enhanced data gained by translation, increasing noise, scaling the relative position and value of the original data.

For the original data, each data group consisted of 12 dimensions collected by 6 sensors (2 dimensions for each sensor). We labeled the data by strictly following the file-naming rule of a-b-c-d.txt. a represents the position of the sensor node, b represents the name of the gesture, c represents the dimension of the sensor data, d represents the collection times. The age and gender distribution of the volunteers and their finger-moving conditions (including moving stability and whether one is left- or right-handed) are shown in Tables 4 and 5.

Table 4. The age and gender distribution of the volunteers.

		Age			
		11~20	21~40	41~60	61+
Gender	Male	1	2	2	1
	Female	0	1	2	1

Table 5. The finger moving condition of the volunteers.

Normal	With some sort of problems or instability	Left handedness
8/10	2/10 (male, 41~60, chronic alcoholic, slightly shaking hands when relaxed; female, 21~40, has wound to the ring finger)	3/10 (Male, 21~40, 41~60; Female, 41~60)

5.2. Comparison of classification algorithms

In this paper, KNN and SVM are selected as comparable classification algorithms with our CNN+NN model, as they conform to our objectives of deploying offline identification algorithm on an MCU-level development platform and then ASIC. These two supervised multiple classifiers are usually designed with hardware description language and then implemented on ASIC in past studies [47–49].

We divided the raw data into two groups - training data and testing data - at the ratio of 2:1 for the three algorithms. We then pinched, cropped, and added stochastic noise to the raw data to obtain 1200 sets of enhanced data. They are split to training and testing data still at the ratio of 2:1. Both raw data and enhanced data are tested for recognition accuracy.

The final results are shown in Table 6. Ideally, the accuracy of the three algorithms tested on raw data is close. However, when data are enhanced in various ways, CNN+NN has a much distinct accuracy at 95.12%, which proves its strong robustness, generalization ability, and suitability for real application scenarios.

Table 6. Comparisons of classifier algorithm.

Classification algorithm	Accuracy of the raw data test	Accuracy of the enhanced data test
KNN	99.09%	89.1%
SVM	98.88%	85.12%
CNN+NN	97.15%	95.12%

5.3. Evaluation of hardware implementation

The performance of an HW accelerator for the MCU can be divided into 3 categories: overall hardware utilization, time performance, and power consumption.

Here, we chose Intel's DE10-Lite development board as the hardware development platform with 10M50DAF484C7G as the main chip. For low power consumption, this design uses a 55 nm CMOS technological process, 49760 LUTs, 1638 Kbit M9K memory, 144 18 × 18 multipliers, and 4 PLLs. Table 7 shows the re-source consumption of this design. It should be noted that the 144 18 × 18 multipliers in the MAX10 are recognized as 288 9 × 9 multipliers in the Quartus software.

Table 7. Hardware resource consumption.

Module	LUTs	Registers	Memory bits	Multiplier elements
Cortex-M0 Kernel	5427	1324	0	6
AHB-Lite BUS	148	46	0	0
APB BUS	125	24	0	0
Peripherals	576	495	16884	4
Preprocessing Module	1580	717	3360	6
Feature Extraction Module	3665	1014	6726	0
Classification Module	2240	2082	4571	16
Total	13769/49760	5702	31541/1667	32/288

We chose a desktop processor, two mobile application processors, and the STM-MCU as the evaluation objects. We used a similar algorithm that was run on the Intel Core i7-9750H, Rockchip RK3399 Pro, Raspberry Pi 3B+, and STM32F407. As multiple groups of detection data were input into the FM to measure the operation time, the average values were recorded. The data for each platform are listed in Table 8.

Table 8. Acceleration performance.

System on Chip	Architecture	Core Number	Frequency	Latency
Proposed	Cortex M0	1	50MHz	1.544ms
Intel i7-9750H	Coffee Lake	6	2.6GHz	0.176ms
Rockchip RK3399 Pro	Cortex A72	6	2.0GHz	1.398ms
Raspberry Pi 3B+	Cortex A53	4	1.4GHz	3.232ms
STM32F407	Cortex M4	1	168MHz	16.683ms

According to data in Table 8, in this specific scenario and in the range of MCUs, the proposed system has a significant performance, as M4 and M3 cannot even complete the calculations. Moreover, as an MCU, the proposed system outperforms some of the SBC. For example, its performance is more than twice as much of Cortex-A53 (which is usually used in Raspberry Pi), and is close to the high-performance of Cortex-A72.

With this astounding performance, referring to the results in Table 7, the resource consumption of the proposed system is only in line with M3. This accomplishment is of great importance for the design of ASIC and offline high-performance computing.

The coupling modes based on different accelerators are shown in Figure 14. Direct coupling to the AHB BUS was compared to that of the peripheral interface. Because the Cortex-M0 was

transplanted to the FPGA through the IP core, only the SPI (Serial Peripheral Interface) and UART (Universal Asynchronous Receiver Transmitter) peripheral interfaces are discussed. Test comparison revealed that delay in the direct coupling and Cortex-M0 is the lowest, which is also determined by the coupling mode. However, for the SPI and UART, the main delay occurs in the data transmission process in the communication protocol of the two. Furthermore, the accelerator would become more universal in the way with a peripheral interface. Low power consumption and less resource consumption are critical factors in designing the accelerator for the MCU.

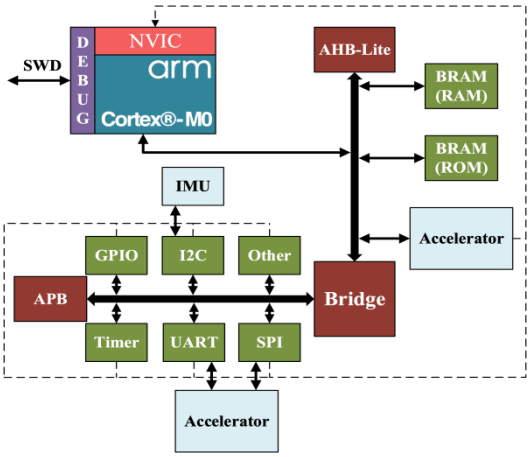


Figure 14. Different couplings of the accelerator.

Since the neural network system architecture is recommended for the application-specific scenario, it is difficult to find the same architecture for comparison. Hence, the power consumption of the CNN and NN, the core part of the accelerator, are compared with those of the other parts (see Table 9). To achieve low power consumption, we adopted methods that minimize the power usage in the design, including the common gated clock and the Gray code encoding for the weight address bit.

Table 9. Comparison of accelerator resource consumption and performance.

		LUTs	Registers	Memory bits	DSP	Power/mW
CNN	Proposed	3665	1014	6726	0	133.68
	[25]	4901	2983	4800	0	380
	[50]	15285	2074	57100	564	607
	[51]	5717	6207	3900	20	370
Multilayer Perceptron NN	Proposed	2240	2082	4571	8	124.48
	[26]	2047	/	/	8	34.91
	[52]	1111	408	4000	6	183.27

Based on these data, the individual operation frequency of the acceleration module can reach 96 MHz after time sequence analysis and constraint operation. This suggests that the shorter lay is obtained as the accelerator combines with a certain buffer through a peripheral coupling. The FPGA in an Ultra-low-power level of Microsemi IGL00 with only 0.21 MW static power consumption has been previously used [26]. However, the static power consumption of MAX10, which is used here, exceeds 90 MW. Due to the restricted condition, the Cyclone 10LP was used to analyze the power consumption of the Cyclone 10LP chip series using the Power Analyzer Tool of the Quartus. Cyclone

10LP is Intel's low-power FPGA, with a static power consumption of 30 MW only. Our proposed design is simulated by the Quartus and achieves a lower power consumption index. Therefore, this series of chips should be considered for further testing and applications.

6. Conclusion

In this paper, we propose a low-cost wearable edge device with a neural network accelerator based on the IMU sensor for gesture recognition. Firstly, the prototype glove is designed to locally collect data, process data, and complete the large volume data calculations off-line. Secondly, with the pre-stage processing module and serial-parallel hybrid method, the device is of low-power and low-latency. As an MCU that consumes at an MCU level, it performs eight times higher than the existing high-performance MCU and outperforms some SBCs. Thirdly, the whole system is a software-hardware co-design that is potentially transferrable to other scenarios. Moreover, a new activation function was designed for the multilayer perception neural network to improve recognition accuracy, and the feature extract process of CNN to complete classification is rather automatic that doesn't need expert supervision in the process. Finally, all the data are open-sourced and will be continuously updated for other researchers for further use.

Our work promotes embedded systems and accessible edge computing models in the field of hand rehabilitation. However, we recognize that our framework presents four core limitations. The first is related to the amount and the type of hand gestures we tested. Since our intention was to make a technical prototype for locally recognizing medical and healing exercises for hands, we used only ten hand gestures for numbers instead of actual movements that hospitals are currently using. We will continuously work with the medical institute for future development. The second is that the whole device falls short in its undesirable size. It's not comfortable enough for long-time wearing or in-the-wild uses. Moreover, the power consumption has not reached its lowest, because we didn't use FPGA devices with the lowest power consumption. Last but not the least, further power analysis and design verification by ASIC tools are needed.

Future studies may focus on the three directions below: (a) Improvement in resource utilization and calculation performance, (b) ASIC implementation of the accelerator with Cortex-M0 processor core via EDA (Electronic Design Automation) tools optimization, (c) More specific and targeted recognition and evaluation schemes for different medical scenarios, such as sign language interpretation, finger movement rehabilitation after stroke and wrist movement evaluation after fracture, etc. Ultimately, we hope our work can be a good reference to improve the device accessibility, usability, versatility for different groups of users.

Acknowledgments

The authors would like to thank volunteers for participating in data collection, and VeriMake Research for providing developing and testing equipment. This research was partly funded by Industry-University-Research Collaboration Foundation of Fuzhou University grant number 0101/01011919.

Conflict of interest

The authors declare there is no conflict of interest.

References

1. C. Wolf, A mathematical model for the propagation of a hantavirus in structured populations, *Discrete Continuous Dyn. Syst. Ser. B*, **4** (2004), 1065–1089.
2. J. Wu, R. Jafari, Orientation Independent Activity/Gesture Recognition Using Wearable Motion Sensors, *IEEE Internet Things J.*, **6** (2018), 1427–1437.
3. P. K. Pisharady, M. Saerbeck, Recent methods and databases in vision-based hand gesture recognition: A review, *Comput. Vis. Image Underst.*, **141** (2015), 152–165.
4. H. S. Hasan, S. Kareem, Human computer interaction for vision based hand gesture recognition: A survey, *Artif. Intell. Rev.*, **43** (2015), 1–54.
5. H. I. Lin, Hsien-I., M. H. Hsu, W.-K. Chen, Human hand gesture recognition using a convolution neural network, In *IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, IEEE, 2014, 1038–1043.
6. O. K. Oyedotun, A. Khashman, Deep learning in vision-based static hand gesture recognition, *Neural Comput. Appl.*, **28** (2017), 3941–3951.
7. Z. Lu, X. Chen, Q. Li, X. Zhang, P. Zhou, A Hand Gesture Recognition Framework and Wearable Gesture-Based Interaction Prototype for Mobile Devices, *IEEE Trans. Hum. Mach. Syst.*, **44** (2014), 293–299.
8. Y. Huang, L. Gao, Y. Zhao, X. Guo, C. Liu, P. Liu, Highly flexible fabric strain sensor based on graphene nanoplatelet-polyaniline nanocomposites for human gesture recognition, *J. Appl. Polymer Sci.*, **134** (2017), 45340.
9. M. Panwar, Hand gesture recognition based on shape parameters, *Int. Conf. Comput., Commun. Appl.*, Dindigul, Tamilnadu, 2012, 1–6.
10. C. Weng, Y. Li, M. Zhang, K. Guo, X. Tang, Z. Pan, Robust Hand Posture Recognition Integrating Multi-cue Hand Tracking. In *Int. Conf. Technol. E-learn. Digital Entertain.*, Springer, Berlin, Heidelberg, 497–508.
11. D. H. Kim, J. Lee, H. S. Yoon, J. Kim, J. Sohn, Vision-based arm gesture recognition for a long-range human–robot interaction, *J. Supercomput.*, **65** (2013), 336–352.
12. J. Li, H. Huai, J. Gao, D. Kong, L. Wang, Spatial-temporal dynamic hand gesture recognition via hybrid deep learning model, *J. Multimodal User Interfaces*, **13** (2019), 363–371.
13. Z. Lu, S. Qin, X. Li, L. Li, D. Zhang, One-shot learning hand gesture recognition based on modified 3d convolutional neural networks, *Mach. Vis. Appl.*, **30** (2019), 1157–1180.
14. A. Sarkar, A. Gepperth, U. Handmann, T. Kopinski, Dynamic Hand Gesture Recognition for Mobile Systems Using Deep LSTM, *Int. Conference Intell. Hum. Comput. Interact*., Springer, Cham, 2017.
15. T. Gonzalez-Sanchez, D. Puig, Real-time body gesture recognition using depth camera, *Electron. Lett.*, **47** (2011), 697–698.
16. S. Diego, F. Bruno, B. Byron, HAGR-D: A Novel Approach for Gesture Recognition with Depth Maps, *Sensors*, **15** (2015), 28646–28664.
17. B. Fang, F. Sun, H. Liu, 3D human gesture capturing and recognition by the IMMU-based data glove, *Neurocomputing*, **277** (2018), 198–207.
18. P. Rouanet, P. Y. Oudeyer, F. Danieau, D. Filliat, The Impact of Human–Robot Interfaces on the Learning of Visual Objects, *IEEE Trans. Robot.*, **29** (2013), 525–541.
19. J. Zhu, L. G. Blumberg, Y. Zhu, M. Nisser, E. L. Carlson, X. Wen, et al., CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form, In *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2020, 1–13.

20. D. Y. Lee, S. H. Lee, I. Oakley, Nailz: Sensing Hand Input with Touch Sensitive Nails, In *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2020, 1–13.
21. A. Akl, C. Feng, S. Valaee, A Novel Accelerometer-Based Gesture Recognition System, *IEEE Trans. Signal Process.*, **59** (2011), 6197–6205.
22. M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, W. Sung, FPGA-based low-power speech recognition with recurrent neural networks, *IEEE Int. Workshop Signal Process. Syst. (SiPS)*, 2016, 230–235.
23. V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, M. Blott, FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs. In *28th Int. Conf. Field Program. Logic Appl. (FPL)*. IEEE, 2018, 89–897.
24. R Xie, J Cao, Accelerometer-Based Hand Gesture Recognition by Neural Network and Similarity Matching, *IEEE Sensors J.*, **16** (2016), 4537–4545.
25. F. Ge, N. Wu, H. Xiao, Y. Zhang, F. Zhou, Compact Convolutional Neural Network Accelerator for IoT Endpoint SoC, *Electronics*, **8** (2019), 497.
26. M. Roukhami, M. T. Lazarescu, F. Gregoretti, Y. Lahbib, A. Mami, Very Low Power Neural Network FPGA Accelerators for Tag-Less Remote Person Identification Using Capacitive Sensors, *IEEE Access*, **7** (2019), 102217–102231.
27. J Chauhan, S Seneviratne, Y Hu, A. Misra, Breathing-Based Authentication on Resource-Constrained IoT Devices using Recurrent Neural Networks, *Computer*, **51** (2018), 60–67.
28. S. Okada, S. Ishibashi, T. Nishida, On-Line Unsupervised Segmentation for Multidimensional Time-Series Data and Application to Spatiotemporal Gesture data, In *Int. Conference Ind. Eng. Other Appl. Applied Intell. Syst.*, IEA/AIE 2010, Proceedings, Part I. DBLP, 2010.
29. O. Dehzangi, V. Sahu, IMU-Based Robust Human Activity Recognition using Feature Analysis, Extraction, and Reduction, *24th Int. Conf. Pattern Recognition (ICPR)*, IEEE, 2018, 1402–1407.
30. J. H. Kim, G. S. Hong, B. G. Kim, D. P. Dogra, deepGesture: Deep learning-based gesture recognition scheme using motion sensors, *Displays*, **55** (2018), 38–45.
31. D. Jeong, B.-G. Kim, S.-Y. Dong, Deep Joint Spatiotemporal Network (DJSTN) for Efficient Facial Expression Recognition, *Sensors*, **20** (2020), 1936.
32. L. García-Hernández, M. Pérez-Ortiz, A. Araúzo-Azofra, L. Salas-Morera, C. Hervás-Martínez, An evolutionary neural system for incorporating expert knowledge into the UA-FLP, *Neurocomputing*, **135** (2014), 69–78.
33. E. Keogh, S. Chu, D. Hart, M. Pazzani, An online algorithm for segmenting time series, *Proc. 2001 IEEE Int. Conf. Data Min.*, IEEE, 2001, 289–296.
34. S. Mallat, Wavelets for a vision, *Proc. IEEE*, **84** (1996), 604–614.
35. L. Perez, J. Wang, The Effectiveness of Data Augmentation in Image Classification using Deep Learning, *arXiv preprint arXiv:1712.04621*.
36. J. Sola, J. Sevilla, Importance of input data normalization for the application of neural networks to complex industrial problems, *IEEE Trans. Nucl. Sci.*, **44** (1997), 1464–1468.
37. J. Han, J. Pei, M. Kamber, Data mining: concepts and techniques. Elsevier, 2011.
38. S. T. Chakradhar, M. Sankaradass, V. Jakkula, S. Cadambi, A dynamically configurable coprocessor for convolutional neural networks, *37th Int. Symp. Comput. Archit. (ISCA 2010)*, ACM, 2010.
39. T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, et al., DianNao: A Small-Footprint

High-Throughput Accelerator for Ubiquitous Machine-Learning, *ACM SIGARCH Comput. Architect. News*, **42** (2014), 269–284.
40. P. Sayyah, T. L. Mihai, S. Bocchio, E. Ebeid, G. Palermo, D. Quaglia, et al., Virtual Platform-Based Design Space Exploration of Power-Efficient Distributed Embedded Applications, *ACM Trans. Embed. Comput. Syst., (TECS)*, **14** (2015), 1–25.
41. J. G. Proakis, D. G. Manolakis, Digital signal processing: principles, algorithms, and applications, 1996.
42. U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, Springer, 2007.
43. J. Cho, S. Mirzaei, J. Oberg, R. Kastner, FPGA-Based Face Detection System Using Haar Classifiers, In *Proc. ACM/SIGDA Int. Symp. Field Program. Gate arrays*, 103–112.
44. J. Hegarty, J. Brunhaver, Z. Devito, J. Ragan-Kelley, N.Cohen, S. Bell, et al., Darkroom: Compiling high-level image processing code into hardware pipelines, *ACM Trans. Graph.*, **33** (2014), 1–11.
45. J. Kim, B. Kim, P. P. Roy, D. Jeong, Efficient facial expression recognition algorithm based on hierarchical deep neural network structure, *IEEE Access*, **7** (2019), 41273–41285.
46. H. Akima, A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures, *J. ACM*, **17** (1970), 589–602.
47. M. Shi, A. Bermak, S. Chandrasekaran, A. Amira, S. Brahim-Belhouari. A Committee Machine Gas Identification System Based on Dynamically Reconfigurable FPGA, *IEEE Sensors J.*, **8** (2008), 403–414.
48. M. Hamouda, H. F. Blanchette, K. Al-Haddad, F. Fnaiech. An Efficient DSP-FPGA-Based Real-Time Implementation Method of SVM Algorithms for an Indirect Matrix Converter, *IEEE Trans. Ind. Electron.*, **58** (2011), 5024–5031.
49. N. Attaran, A. Puranik, J. Brooks, T. Mohsenin, Embedded Low-Power Processor for Personalized Stress Detection, *IEEE Trans. Circuits Syst. II-Express Briefs*, **65** (2018), 2032–2036.
50. Z. Li, L. Wang, S. Guo, Y. Deng, Q. Dou, H. Zhou, et al., Laius: An 8-bit fixed-point CNN hardware inference engine, In *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. and 2017 IEEE Int. Conf. Ubiquitous Computi. Commun. (ISPA/IUCC)*, 143–150.
51. Y. Zhang, N. Wu, F. Zhou, M. R. Yahya, Design of Multifunctional Convolutional Neural Network Accelerator for IoT Endpoint SoC, *In Proc. World Congress Eng. Comput. Sci. 2018*, 16–19.
52. C. M. Morales, U. Flores, M. A. Medina, M. D. Saazar, J. A. Caballero, D. C. Cruz, et al., Digital Artificial Neural Network Implementation on a FPGA for data classification, *IEEE Latin Am. Trans.*, **13** (2015), 3216–3220.

AIMS Press